Lab Exercise: Understanding Static

INTERTECH

Instructors Who Consult. | Consultants Who Teach.

About Intertech

Thank you for choosing Intertech for your training from Udemy. The next page of this document will get you started with your lab if you'd like to skip ahead. Below is a brief overview of Intertech as well as a promo code for you for future live Intertech trainings.

Intertech (www.intertech.com) is the largest combined software developer **training** and **consulting** firm in Minnesota. Our unique blend of training, consulting, and mentoring has empowered technology teams in small businesses, Fortune 500 corporations and government agencies since 1991.

Our training organization offers live in-classroom and online deliveries, private on-site deliveries, and on-demand options such as the course in which you've enrolled from Udemy. We cover a broad spectrum of .NET, Java, Agile/Scrum, Web Development, and Mobile technologies. See more information on our training and search for courses by **clicking here**.

As a Udemy customer, you can save on any live in-classroom, live online, or onsite training with Intertech as well. Just use promo code "Udemy_Labs" when enrolling **and** save 35% on the course price.

We appreciate you choosing Intertech on Udemy for this course!

Lab Exercise

Understanding Static

As you learned in class, data can be stored in objects but can also be associated with the class itself. Data associated with the class is held in static variables. Likewise, methods that operate on static data or provide functionality to the class are called static methods.

In this lab, you explore the addition of static variables and methods to a class.

Specifically, in this lab you will:

- Explore a prewritten class or type called Order
- Add a static variable to an Order class
- Add several static methods to Order
- Add an instance method to use the static data in Order
- Optionally add a static initialization block
- Optionally explore an additional constructor on Order and the use of the *this* keyword.

Scenario

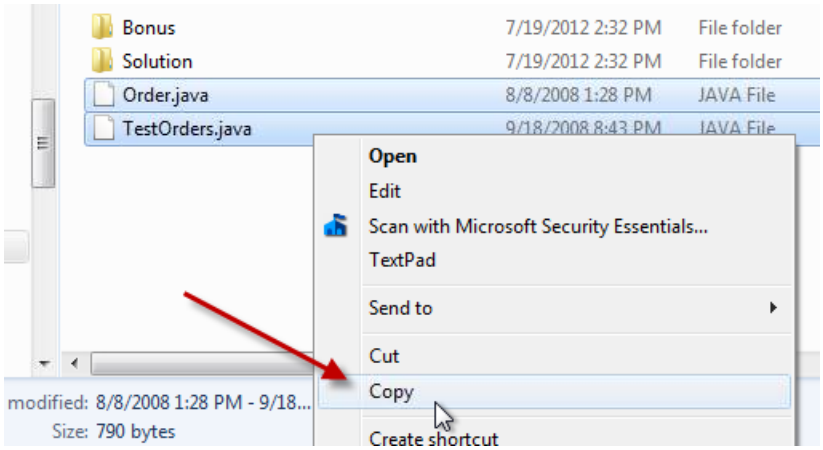
The Acme Order System is off to great start. In this lab, you help add code to an Order class to deal with taxes. Yes, even Acme needs to take care of its fair share of the tax burden. Although the tax rate may change from year to year, the same tax rate applies to all current orders. This sounds like a perfect place to apply a static variable.

Step 1: Explore the Order class

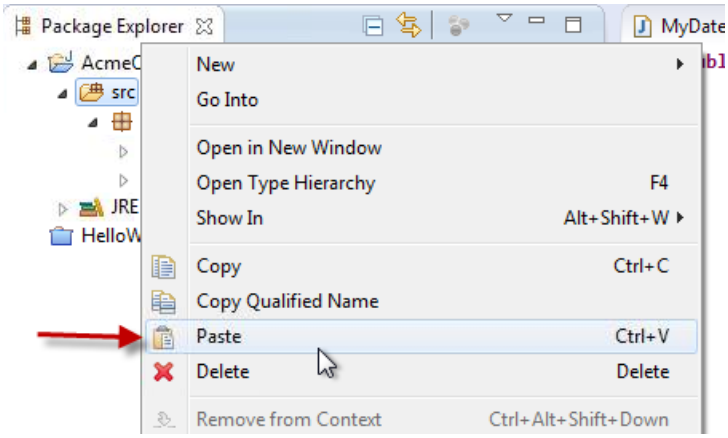
An Order class and a TestOrders class have already been created for you to use in this lab. The first step of this lab is to explore and understand the Order class and see TestOrders create and display some Order objects.

1.1 Import Order.java and TestOrders.java from the IntertechLearnJava zip (found in the Resources section for the "HelloWorld Lab").

1.1.1 Right-click on the files, and request a copy of the files.



1.1.2 Back in the Eclipse IDE, right-click on the src folder in the AcmeOrderSystem project, and request to paste the copied file into the project.



1.2 Double-click on both the Order.java and TestOrders.java files in the Package Explorer view to open each of the files in an editor. Explore both classes. The Order class represents an order in the Acme System, while the TestOrders class allows you to test the functionality of the Order class.

1.2.1 What type of instance variables does the Order class have?

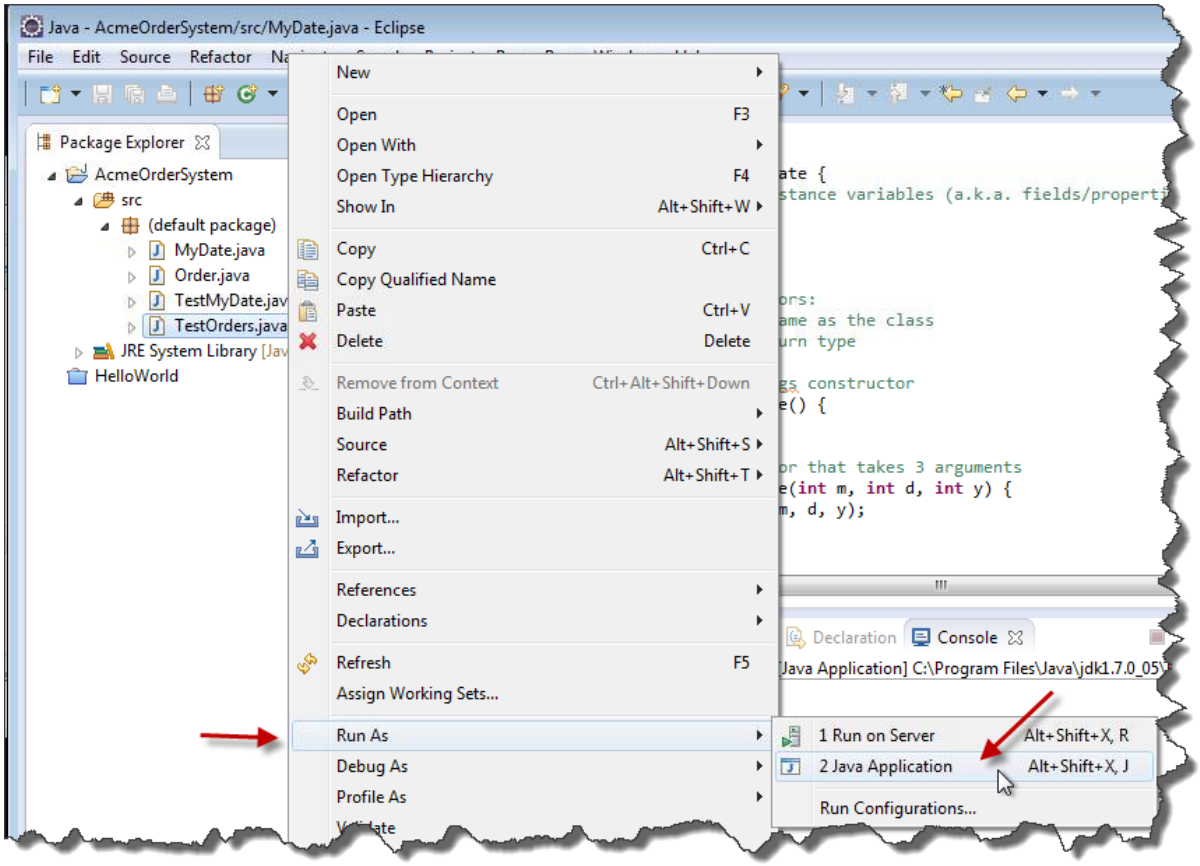
1.2.2 How is your MyDate class reused?

1.2.3 How do you construct an Order?

1.2.4 How many Order objects are created in the main method of TestOrders?

1.3 Run the TestOrders class.

1.3.1 Right-click on TestOrders, and select *Run As > Java Application*.



1.3.2 When you run the test, the output in the Console view should look like the following. If it looks different, go back and try to fix it. Ask for help if you get stuck.

```
10 ea. Anvil for Wile E Coyote
125 ea. Balloon for Bugs Bunny
```

Step 2: Add a static variable to Order

In order to compute the tax on any Order, the tax rate must be in the system. The tax rate should be applied equally to all orders, making a static variable on Order well suited for this data.

2.1 Add a static variable, taxRate, to the Order class, and initialize the tax rate to 5%.

```
static double taxRate = 0.05;
```

2.2 Add a static method to the Order class to set a new taxRate.

```
public static void setTaxRate(double newRate) {
  taxRate = newRate;
}
```

2.3 Add another static method to compute the tax on an amount provided as a parameter.

```
public static void computeTaxOn(double anAmount){
  System.out.println("The tax for " + anAmount + " is: " +
anAmount * Order.taxRate);
}
```

2.4 Add a new method to compute the tax for an Order object.

```
public double computeTax() {
  System.out.println("The tax for this order is: " + orderAmount
* Order.taxRate);
  return orderAmount * Order.taxRate;
}
```

Why should these last two methods be static methods, that is, have the *static* modifier?

Step 3: Add code to test the static variable

3.1 Add code to TestOrders to test the new static variable and methods. Below the existing lines of code in the main method, add the following code:

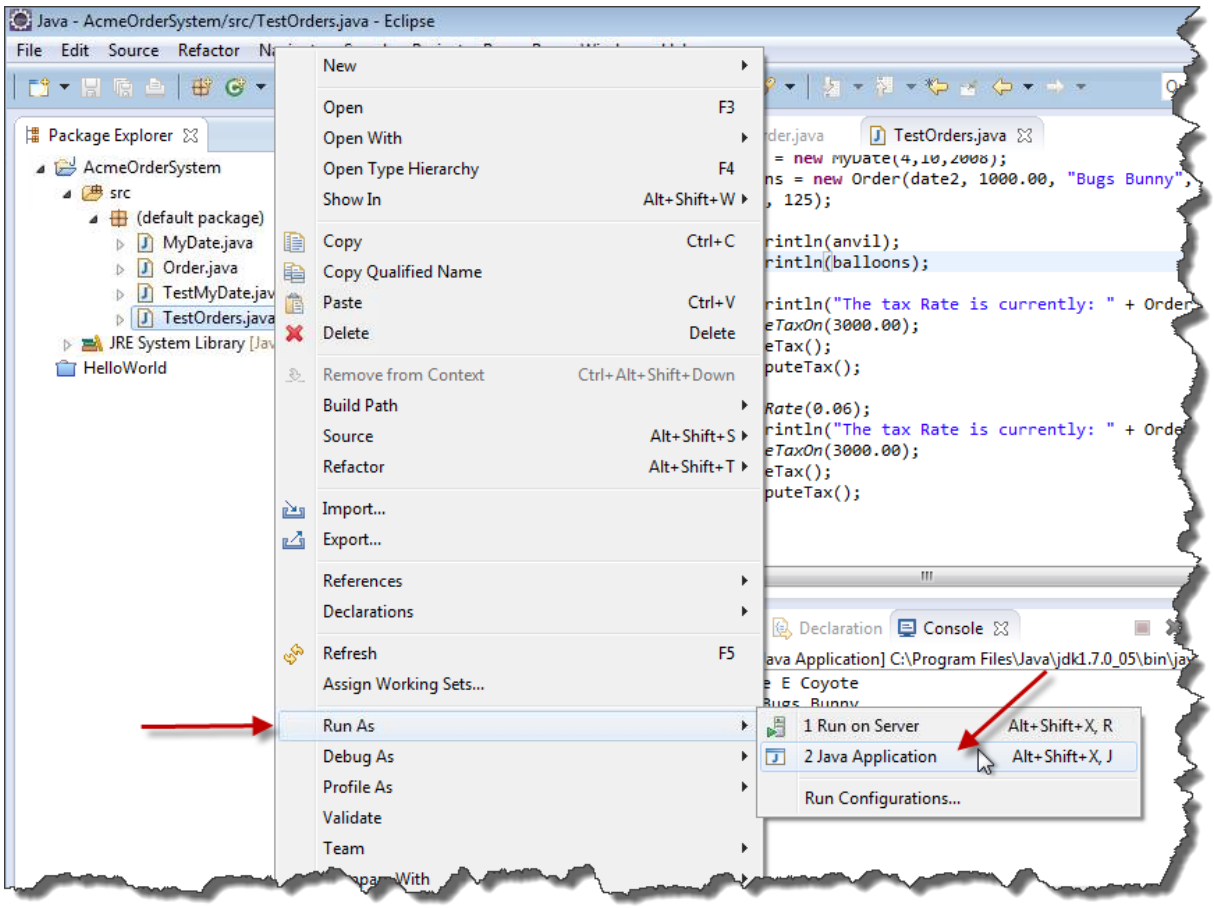
```
System.out.println("The tax Rate is currently: " +
Order.taxRate);
Order.computeTaxOn(3000.00);
anvil.computeTax();
balloons.computeTax();

Order.setTaxRate(0.06);
System.out.println("The tax Rate is currently: " +
Order.taxRate);
Order.computeTaxOn(3000.00);
anvil.computeTax();
balloons.computeTax();
```

3.2 Save your files, and fix any errors in Order and the TestOrders file.

Step 4: Run TestOrders

4.1 Right-click on TestOrders, and select ***Run As > Java Application***.



4.2 When you run the test, the output in the Console view should look like the following. If it looks different, go back and try to fix it. Ask for help if you get stuck.

```
10 ea. Anvil for Wile E Coyote
125 ea. Balloon for Bugs Bunny
The tax Rate is currently: 0.05
The tax for 3000.0 is: 150.0
The tax for this order is: 100.0
The tax for this order is: 50.0
```

```
The tax Rate is currently: 0.06
The tax for 3000.0 is: 180.0
The tax for this order is: 120.0
The tax for this order is: 60.0
```

Lab Solution

Order.java

```
public class Order {
  MyDate orderDate;
  double orderAmount = 0.00;
  String customer;
  String product;
  int quantity;

  static double taxRate = 0.05;

  public static void setTaxRate(double newRate) {
    taxRate = newRate;
  }

  public static void computeTaxOn(double anAmount) {
    System.out.println("The tax for " + anAmount + " is: " +
anAmount
        * Order.taxRate);
  }

  public Order(MyDate d, double amt, String c, String p, int q)
{
    orderDate = d;
    orderAmount = amt;
    customer = c;
    product = p;
    quantity = q;
  }

  public String toString() {
    return quantity + " ea. " + product + " for " + customer;
  }

  public double computeTax() {
    System.out.println("The tax for this order is: " +
orderAmount
        * Order.taxRate);
    return orderAmount * Order.taxRate;
  }
}
```

TestOrders.java

```
public class TestOrders {

  public static void main(String[] args) {
    MyDate date1 = new MyDate(1,20,2008);
    Order anvil = new Order(date1, 2000.00, "Wile E Coyote",
"Anvil",
       10);

    MyDate date2 = new MyDate(4,10,2008);
    Order balloons = new Order(date2, 1000.00, "Bugs Bunny",
       "Balloon", 125);

    System.out.println(anvil);
    System.out.println(balloons);

    System.out.println("The tax Rate is currently: " +
Order.taxRate);
    Order.computeTaxOn(3000.00);
    anvil.computeTax();
    balloons.computeTax();

    Order.setTaxRate(0.06);
    System.out.println("The tax Rate is currently: " +
Order.taxRate);
    Order.computeTaxOn(3000.00);
    anvil.computeTax();
    balloons.computeTax();
  }
}
```

Bonus Lab

Step 5: Add a static initialization block

Currently, the static tax rate is initialized when declared. Remove the initialization from the tax rate declaration (as shown in the line of code below), and then add a static initialization block to do this same work. This change should not affect the output of the application TestOrders.

```
static double taxRate;
```

Step 6: Add a constructor and use "this"

Add a new constructor to the Order class that assumes the caller wants to order a single "anvil," the most popular product.

```
public Order(MyDate d, double amt, String c){
  orderDate = d;
  orderAmount = amt;
  customer = c;
  product = "Anvil";
  quantity = 1;
}
```

Notice that much of the code between this new constructor and the existing constructor is duplicated. Can you reuse the original constructor code and reduce the code in this new constructor by making use of the *this* keyword? Create an additional test in TestOrders as shown below to check your work.

```
MyDate date3 = new MyDate(5, 20, 2008);
Order anotherAnvil = new Order(date3, 200, "Road Runner");
System.out.println(anotherAnvil);
```

Give Intertech a call at **1.800.866.9884** or visit Intertech's website.